

Data wrangling with dplyr

Ronny A. Hernández Mora.

 @RonnyHdezM

 ronnyhdez

 <http://ronnyhdez.rbind.io/>

Workshop materials

README.md

¡Bienvenido al curso de ciencia de datos con R!

Este es un curso libre y gratis que puede usar para dar sus primeros pasos con el lenguaje de programación R.



Artwork by @allison_horst

Materiales del curso

Sesión	Presentación	Video
0-Preparación	No hay	https://www.youtube.com/watch?v=NCvJwJSMq60
1- Introducción a las herramientas	Por subir	Por subir



https://github.com/ronnyhdez/curso_ciencia_datos_r

main 4 branches 0 tags

Go to file Add file **Code**

ronnyhdez Merge pull request #11 from ronnyhdez/T8

- img Ref #1 estructura del curso
- presentaciones Ref #8 material presentacion
- sesion_01 Ref 1 material sesion 1
- sesion_02 Ref #8 orden en script segunda ses
- .gitignore Ref #2 materiales sesion 2

Clone

HTTPS SSH GitHub CLI

https://github.com/ronnyhdez/curso_cie

Use Git or checkout with SVN using the web URL.

Download ZIP

10 days ago

What do we want from today's session?

- How to import my data files
- Understand tidy data
- How to use dplyr verbs to deal with my data





Import data

Import data cheat sheet

Data Import :: CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file

write_csv(**x**, **path**, **na** = "NA", **append** = FALSE, **col_names** = **lappend**)

File with arbitrary delimiter

write_delim(**x**, **path**, **delim** = " ", **na** = "NA", **append** = FALSE, **col_names** = **lappend**)

CSV for excel

write_excel_csv(**x**, **path**, **na** = "NA", **append** = FALSE, **col_names** = **lappend**)

String to file

write_file(**x**, **path**, **append** = FALSE)

String vector to file, one element per line

write_lines(**x**, **path**, **na** = "NA", **append** = FALSE)

Object to RDS file

write_rds(**x**, **path**, **compress** = c("none", "gz", "bzip2", "xz", ...))

Tab delimited files

write_tsv(**x**, **path**, **na** = "NA", **append** = FALSE, **col_names** = **lappend**)

Read Tabular Data

- These functions share the common arguments:

read_*(**file**, **col_names** = TRUE, **col_types** = NULL, **locale** = default_locale(), **na** = c("", "NA"), **quoted_na** = TRUE, **comment** = "", **trim_ws** = TRUE, **skip** = 0, **n_max** = Inf, **guess_max** = min(1000, **n_max**), **progress** = interactive())

a,b,c
1,2,3
4,5,NA

Comma Delimited Files

read_csv("file.csv")

To make file.csv run:

write_file(**x** = "a,b,c(n1,2,3)n4,5,NA", **path** = "file.csv")

a,b,c
1,2,3
4,5,NA

Semi-colon Delimited Files

read_csv2("file2.csv")

write_file(**x** = "a;b;c(n1;2;3)n4;5;NA", **path** = "file2.csv")

a|b|c
1|2|3
4|5|NA

Files with Any Delimiter

read_delim("file.txt", **delim** = "|")

write_file(**x** = "a|b|c(n1|2|3)n4|5|NA", **path** = "file.txt")

a b c
1 2 3
4 5 NA

Fixed Width Files

read_fwf("file.fwf", **col_positions** = c(1, 3, 5))

write_file(**x** = "a b c(n1 2 3)n4 5 NA", **path** = "file.fwf")

a b c
1 2 3
4 5 NA

Tab Delimited Files

read_tsv("file.tsv") Also **read_table**()

write_file(**x** = "a|b|c(n1|2|3)n4|5|NA", **path** = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file

write_file("a,b,c(n1,2,3)n4,5,NA","file.csv")

f <- "file.csv"

A B C
1 2 3
4 5 NA

No header

read_csv(**f**, **col_names** = FALSE)

x y z
A B C
1 2 3
4 5 NA

Provide header

read_csv(**f**, **col_names** = c("x", "y", "z"))

Skip lines

read_csv(**f**, **skip** = 1)

Read in a subset

read_csv(**f**, **n_max** = 1)

Missing Values

read_csv(**f**, **na** = c("1", "m"))

Read Non-Tabular Data

Read a file into a single string

read_file(**file**, **locale** = default_locale())

Read each line into its own string

read_lines(**file**, **skip** = 0, **n_max** = -1L, **na** = character(), **locale** = default_locale(), **progress** = interactive())

Read Apache style log files

read_log(**file**, **col_names** = FALSE, **col_types** = NULL, **skip** = 0, **n_max** = -1, **progress** = interactive())

Read a file into a raw vector

read_file_raw(**file**)

Read each line into a raw vector

read_lines_raw(**file**, **skip** = 0, **n_max** = -1L, **progress** = interactive())

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.

x <- **read_csv**("file.csv"); **problems(x)**

2. Use a **col_*** function to guide parsing.

- **col_guess()** the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime**(**format** = "%Y-%m-%d") Also **col_date**(**format** = "%Y-%m-%d", **col_time**(**format** = "%Y-%m-%d %H:%M:%S")
- **col_factor**(**levels**, **ordered** = FALSE)
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

x <- **read_csv**("file.csv", **col_types** = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))

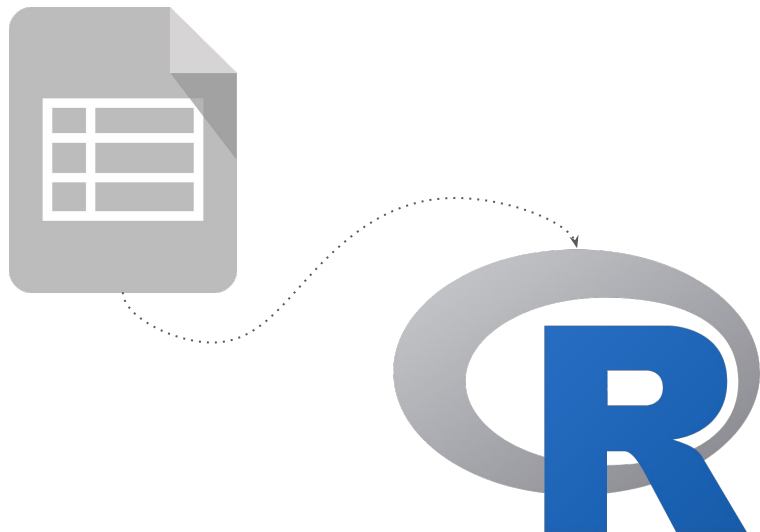
3. Else, read in as character vectors then parse with a **parse_*** function.

- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**
- **xSA** <- **parse_number**(**xSA**)



How to read a data file in my R session?

```
read_csv( "path_to_file" )
```



```
read_*( )
```



<https://readr.tidyverse.org/>



<https://readxl.tidyverse.org/>



break 12:05 a 12:15

dplyr : go wrangling



Domando datos con dplyr

Dplyr cheat sheet

Data Transformation with dplyr : CHEAT SHEET



dplyr functions work with pipes and expect tidy data. In tidy data:



Each **variable** is in its own **column**



Each **observation**, or **case**, is in its own **row**



$x \%>\% f(y)$ becomes $f(x, y)$



pipes

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

`summarise(data, ...)`
Compute table of summaries.
`summarise(mtcars, avg = mean(mpg))`

`count(x, ..., wt = NULL, sort = FALSE)`
Count number of rows in each group defined by the variables in ... Also `tally()`.
`count(mtcars, cyl)`

Group Cases

Use `group_by(data, ..., add = FALSE)` to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

`mtcars %>%
 group_by(cyl) %>%
 summarise(avg = mean(mpg))`

Use `rowwise(data, ...)` to group data into individual rows. dplyr functions will compute results for each row. Also used to apply functions to list-columns without purrr functions.

`starwars %>%
 rowwise() %>%
 mutate(film_count = length(films))`

`ungroup(x, ...)` Returns ungrouped copy of table.
`ungroup(g_mtcars)`

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

`filter(data, ...)` Extract rows that meet logical criteria.
`filter(mtcars, mpg > 20)`

`distinct(data, ..., keep_all = FALSE)` Remove rows with duplicate values.
`distinct(mtcars, gear)`

`slice(data, ...)` Select rows by position.
`slice(mtcars, 10:15)`

`slice_sample(data, ..., n, prop, weight_by = NULL, replace = FALSE)` Randomly select rows. Use `n` to select a number of rows and `prop` to select a fraction of rows.
`slice_sample(mtcars, n = 5, replace = TRUE)`

`slice_min(data, order_by, ..., n, prop, ties = TRUE)` and `slice_max()` Select rows with the lowest and highest values.
`slice_min(mtcars, mpg, prop = 0.25)`

`slice_head(data, ..., n, prop)` and `slice_tail()` Select the first or last rows.
`slice_head(mtcars, n = 5)`

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%		xor()
>	>=	is.na()	!	&	

See ?base::Logic and ?Comparison for help.

ARRANGE CASES

`arrange(data, ...)` Order rows by values of a column or columns (low to high), use with `desc()` to order from high to low.
`arrange(mtcars, mpg)`
`arrange(mtcars, desc(mpg))`

ADD CASES

`add_row(data, ..., before = NULL, after = NULL)`
Add one or more rows to a table.
`add_row(cars, speed = 1, dist = 1)`

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

`pull(data, var = 1)` Extract column values as a vector. Choose by name or index.
`pull(mtcars, wt)`

`select(data, ...)` Extract columns as a table. Also `select_if()`.
`select(mtcars, mpg, wt)`

`relocate(data, ..., before = NULL, after = NULL)`
Move columns to new position.
`relocate(mtcars, mpg, cyl, after = last_col())`

Use these helpers with select() and across()

e.g. `select(mtcars, mpg:cyl)`

<code>contains(match)</code>	<code>num_range(prefix, range)</code>	z, e.g. <code>mpg:cyl</code>
<code>ends_with(match)</code>	<code>one_of(...)</code>	-, e.g. <code>-gear</code>
<code>matches(match)</code>	<code>starts_with(match)</code>	<code>everything()</code>

MANIPULATE MULTIPLE VARIABLES AT ONCE

`across(cols, funs)` Summarise or mutate multiple columns in the same way.
`summarise(mtcars, across(everything(), mean))`

`c_across(cols)` Compute across columns in row-wise data.
`transmute(rowwise(UKgas), n = sum(c_across(1:2)))`

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

`mutate(data, ..., before = NULL, after = NULL)`
Compute new column(s). Also `add_count()`, `add_count()`, and `add_tally()`.
`mutate(mtcars, gpm = 1/mpg)`

`transmute(data, ...)` Compute new column(s), drop others.
`transmute(mtcars, gpm = 1/mpg)`

`rename(data, ...)` Rename columns.
`rename(cars, distance = dist)`



RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes (package = c("dplyr", "tidbly")) • dplyr 1.0.6 • tidbly 3.1.2 • Updated: 2021-06

`select()`

`filter()`

`mutate()`

`group_by()`

`summarise()`

primera_instruccion %>%

segunda_instruccion %>%

tercera_instruccion





Supplementary text

The R Series

R Markdown

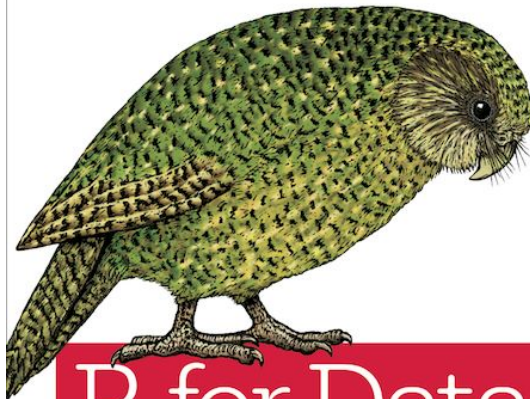
The Definitive Guide



Yihui Xie
J. J. Allaire
Garrett Grolemund

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

O'REILLY®



R for Data Science

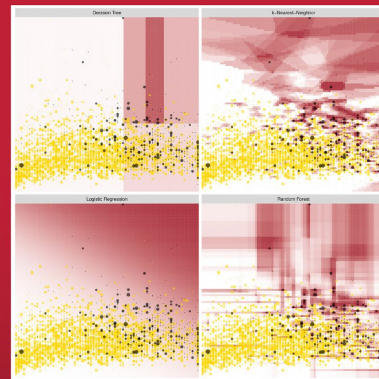
VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

Hadley Wickham &
Garrett Grolemund

Texts in Statistical Science

Modern Data Science with R

Second Edition



Benjamin S. Baumer
Daniel T. Kaplan
Nicholas J. Horton

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

<https://bookdown.org/yihui/rmarkdown/>


<https://es.r4ds.hadley.nz/>


<https://es.r4ds.hadley.nz/>



¡Gracias!

Ronny A. Hernández Mora.

 @RonnyHdezM

 ronnyhdez

 <http://ronnyhdez.rbind.io/>

ronny.hernandezm@gmail.com